SERVLETS - WRITING FILTERS

Servlet Filters are Java classes that can be used in Servlet Programming for the following purposes:

- To intercept requests from a client before they access a resource at back end.
- To manipulate responses from server before they are sent back to the client.

There are are various types of filters suggested by the specifications:

- Authentication Filters.
- Data compression Filters.
- Encryption Filters.
- Filters that trigger resource access events.
- Image Conversion Filters.
- Logging and Auditing Filters.
- MIME-TYPE Chain Filters.
- Tokenizing Filters .
- XSL/T Filters That Transform XML Content.

Filters are deployed in the deployment descriptor file **web.xml** and then map to either servlet names or URL patterns in your application's deployment descriptor.

When the web container starts up your web application, it creates an instance of each filter that you have declared in the deployment descriptor. The filters execute in the order that they are declared in the deployment descriptor.

Servlet Filter Methods:

A filter is simply a Java class that implements the javax.servlet.Filter interface. The javax.servlet.Filter interface defines three methods:

S.N.	Method & Description
1	**public void doFilter (ServletRequest, ServletResponse, FilterChain)** This method is called by the container each time a request/response pair is passed through the chain due to a client request for a resource at the end of the chain.
2	**public void init(FilterConfig filterConfig)** This method is called by the web container to indicate to a filter that it is being placed into service.
3	**public void destroy()** This method is called by the web container to indicate to a filter that it is being taken out of service.

Servlet Filter Example:

Following is the Servlet Filter Example that would print the clients IP address and current date time. This example would give you basic understanding of Servlet Filter, but you can write more sophisticated filter applications using the same concept:

```
// Import required java libraries
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;

// Implements Filter class
public class LogFilter implements Filter  {
   public void  init(FilterConfig config)
                        throws ServletException{
      // Get init parameter
      String testParam = config.getInitParameter("test-param");

      //Print the init parameter
      System.out.println("Test Param: " + testParam);
   }
   public void  doFilter(ServletRequest request,
                 ServletResponse response,
                 FilterChain chain)
                 throws java.io.IOException, ServletException {

      // Get the IP address of client machine.
      String ipAddress = request.getRemoteAddr();

      // Log the IP address and current timestamp.
      System.out.println("IP "+ ipAddress + ", Time "
                                     + new Date().toString());

      // Pass request back down the filter chain
      chain.doFilter(request,response);
   }
   public void destroy( ){
      /* Called before the Filter instance is removed
      from service by the web container*/
   }
}
```

Compile **LogFilter.java** in usual way and put your class file in <Tomcat-installation-directory>/webapps/ROOT/WEB-INF/classes.

Servlet Filter Mapping in Web.xml:

Filters are defined and then mapped to a URL or Servlet, in much the same way as Servlet is defined and then mapped to a URL pattern. Create the following entry for filter tag in the deployment descriptor file **web.xml**

```
<filter>
   <filter-name>LogFilter</filter-name>
   <filter-class>LogFilter</filter-class>
   <init-param>
   <param-name>test-param</param-name>
   <param-value>Initialization Paramter</param-value>
   </init-param>
</filter>
<filter-mapping>
   <filter-name>LogFilter</filter-name>
   <url-pattern>/*</url-pattern>
</filter-mapping>
```

The above filter would apply to all the servlets because we specified **/*** in our configuration. You can specicy a particular servlet path if you want to apply filter on few servlets only.

Now try to call any servlet in usual way and you would see generated log in your web server log. You can use Log4J

logger to log above log in a separate file.

Using Multiple Filters:

Your web application may define several different filters with a specific purpose. Consider, you define two filters *AuthenFilter* and *LogFilter*. Rest of the process would remain as explained above except you need to create a different mapping as mentioned below:

```
<filter>
   <filter-name>LogFilter</filter-name>
   <filter-class>LogFilter</filter-class>
   <init-param>
   <param-name>test-param</param-name>
   <param-value>Initialization Paramter</param-value>
   </init-param>
</filter>

<filter>
   <filter-name>AuthenFilter</filter-name>
   <filter-class>AuthenFilter</filter-class>
   <init-param>
   <param-name>test-param</param-name>
   <param-value>Initialization Paramter</param-value>
   </init-param>
</filter>

<filter-mapping>
   <filter-name>LogFilter</filter-name>
   <url-pattern>/*</url-pattern>
</filter-mapping>

<filter-mapping>
   <filter-name>AuthenFilter</filter-name>
   <url-pattern>/*</url-pattern>
</filter-mapping>
```

Filters Application Order:

The order of filter-mapping elements in web.xml determines the order in which the web container applies the filter to the servlet. To reverse the order of the filter, you just need to reverse the filter-mapping elements in the web.xml file.

For example, above example would apply LogFilter first and then it would apply AuthenFilter to any servlet but the following example would reverse the order:

```
<filter-mapping>
   <filter-name>AuthenFilter</filter-name>
   <url-pattern>/*</url-pattern>
</filter-mapping>

<filter-mapping>
   <filter-name>LogFilter</filter-name>
   <url-pattern>/*</url-pattern>
</filter-mapping>
```